GAINE - A Portable Framework for the Development of Edutainment Applications Based on Multitouch and Tangible Interaction

# GAINE - A Portable Framework for the Development of Edutainment Applications Based on Multitouch and Tangible Interaction

*Andrea Bottino, Andrea Martina, Francesco Strada, Amirhosein Toosi*
*Dipartimento di Automatica e Informatica*
*Politecnico di Torino, Torino, Italy*

***Abstract*— In the last few years, Multitouch and Tangible User Interfaces have emerged as a powerful tool to integrate interactive surfaces and responsive spaces that embody digital information. Besides providing a natural interaction with digital contents, they allow the interaction of multiple users at the same time, thus promoting collaborative activities and information sharing. In particular, these characteristics have opened new exploration possibilities in the edutainment context, as witnessed by the many applications successfully developed in different areas, from children's collaborative learning to interactive storytelling, cultural heritage and medical therapy support. However, due to the availability of different multitouch and tangible interaction technologies and of different target computing platforms, the development and deployment of such applications can be challenging. To this end, in this paper we present GAINE (tanGible Augmented INteraction for Edutainment), a software framework that enables rapid prototyping and development of tangible augmented applications for edutainment purposes. GAINE has two main features. First, it offers developers high-level context specific constructs that significantly reduces the implementation burden. Second, the framework is portable on different operating systems and offers independence from the underlying hardware and tracking technology. In this paper, we also discuss several case studies to show the effectiveness of GAINE in simplifying the development of entertainment and edutainment applications based on multitouch and tangible interaction.**

***Keywords—Multitouch Interfaces, Tangible User Interfaces, Edutainment, Entertainment, portable framework.***

## I. INTRODUCTION

The integration of Multitouch and Tangible User Interfaces is a powerful approach to improve the way users interact with digital information. Multitouch Interfaces (MTIs) provide a direct and more natural interaction with on-screen contents through the use of multiple fingers and gestures. Tangible User Interfaces (TUIs) enable the interaction with digital information through the manipulation of physical objects of the real world [50]. The integration of these two approaches offers a seamless information representation and interaction that spans both digital and physical worlds [48]. This is particularly relevant in edutainment contexts for enhancing learning and discovery activities, especially in applications featuring shared displays, which allow the interaction of more than one user at a time and, thus, promote collaboration, information sharing and the rise of social experiences.

A body of literature explores the opportunities offered by MTIs and TUIs to edutainment and entertainment in various fields, from classroom collaborative learning [12][40][2] to children's decision making and cooperation support [8], interactive storytelling [13][14], pervasive games [15] and museum exhibits [16][17]. While most of these approaches rely on large interactive displays, commercial MTI-TUI games developed for mobile devices are becoming increasingly available [49][51]. A large variety of works have started exploiting these technologies to improve the accessibility to edutainment and entertainment applications of different categories of users, like children and the elderly [39] or people with physical or psychological disabilities. For instance, interactive tabletops games have been used to teach, improve and exercise social and communicational skills among children with autism spectrum disorders [5][6][7]. The potentialities of such games in supporting medical therapies have been also explored. Examples can be found in [10], which describes a game developed for children with cerebral palsy, or in [11], where Virtual Reality training based on a tabletop game was found to be a viable adjunct to conventional physical therapy in facilitating motor learning in patients with traumatic brain injuries.

Despite the capabilities offered by interactive displays and tangible objects, developing applications that fruitfully exploit them presents some inherent difficulties and requires complex technical and programming skills. To this end, we have designed and implemented a software framework called GAINE (*tanGible Augmented INteraction for Edutainment*), aimed at simplifying the prototyping and development of interactive applications in educational and entertainment contexts, which integrates both MTI and TUI approaches. The contribution of our work is threefold:

- providing an abstraction layer that hides the complexity of the underlying hardware architecture and software libraries necessary to manage a tangible augmented interactive game;
- supporting a “write once – deploy everywhere” development model, which enable the portability of the application on a variety of platforms, from large interactive tables to mobile devices like tablets;
- offering developers advanced features and a large set of high level constructs, which can be easily integrated for the rapid development of edutainment applications.

While GAINE shares some characteristics with other toolkits presented in the literature, such as [19][25][27][36], our framework further simplifies the application design by offering developers a reusable, semi complete application that can be totally customized through an XML-style scripting language. The effectiveness of using GAINE in the design and development process is demonstrated through several case studies discussed in the paper.

A preliminary version of this work has been presented in [41]. The main extensions introduced in this paper are the following:

- the original framework was designed for interactive tables only and could be ported only on the main desktop operative systems (Windows, Linux, MacOs); here, we integrated GAINE into the Unity 3D environment to target other computing platforms (i.e., devices running iOS or Android);

- we extended the range of tangibles that can be used by the framework by designing an extensible low-level interaction layer capable of managing MTI and TUI data acquired with different technologies;
- we increased the functionalities offered by GAINE and, thus, the opportunities offered to developers to design novel and compelling applications.

The rest of the paper is organized as follows. In Section II we review the current state of the art related to the recognition and tracking of tangible widgets and to the toolkits supporting tangible-augmented interaction. In Section III we describe the GAINE framework and Section IV and Appendix discuss the implementation, on different platforms, of several applications using GAINE. Finally, conclusions and future works are outlined in Section V.

## II. Related Works

### A. *Object tracking for tangible interaction*

The problem of using physical objects to control or manipulate virtual items on big interactive touch screens has been initially tackled using optical technologies [18]. Recent studies demonstrated the possibility of porting this specific user interaction to other devices relying on capacitive or magnetic tracking techniques. These works are particularly interesting since they allow extending TUIs to portable screens like tablets (e.g., iPads or Android based devices).

With optical techniques, tangibles objects are tagged with fiducials and both touch and fiducial recognition rely on an infrared light (IF) illuminating the table surface and being reflected towards an IF-camera when fingers or fiducials hits the surface. The images of the IF camera are then processed to extract interaction data. Tabletop illumination can use either Diffused Illumination (DI) or Frustrated Total Internal Reflection (FTIR) techniques. With DI, the table surface is illuminated from IF emitters placed below the surface [21], while with FTIR [22] the lights come from the side and is transmitted into the surface material, which is usually a special Enlighten acrylic.

Since tablets and other capacitive displays cannot rely on vision-based tracking, a vast majority of researches have directly exploited the touch screen capacitive sensing technology whilst others have explored the use of magnetic tracking. Most of the capacitive widgets discussed in the literature are passive, i.e. they need human contact to be sensed. CapWidgets [43] are little aluminum knobs with two conductive touch points on their base working as markers. Gestures with the knob, like rotation or placement in hot spots, can be translated into specific actions within the application. Capstones and Zebra Widgets [42] are based on a similar approach. Top and bottom surfaces of Capstones are equipped with 2x2 or 3x3 matrices of conductive markers and the number of active markers sensed by the screen can be changed by stacking Capstones one on top of the other. The 3x3 layout also enables to detect the tangible orientation and two unique IDs. Zebra Widgets are a slider and a knob, which are characterized by two different marker patterns that enable as well the detection of rotation and sliding gestures. TUIC [48] uses a combination of passive material and active modulation circuits to create a hybrid of spatial and frequency tags. Spatial tags use three passive markers for

conveying position information. Frequency tags use a single marker per widget connected to an active circuit, which delivers contact at a given frequency. While allowing a hand-free use and requiring a single contact point, the time encoding of the information increases the latency of marker data and the tangible requires an external power supply. Passive Untouched Capacitive widgets (PUC, [47]), are another type of active widget. PUCs ground themselves by electrically connecting multiple active and inactive intersections of the touch screen grid. While the active intersection is scanned, the inactive intersection serves as ground and the active one is sensed as a touch. The obvious shortcoming is that PUCs require a larger size compared to other passive markers.

A different approach is the one based on magnetic sensing. Gauss Bricks and Gauss Stones [44] [45] are basic blocks composed by two magnets enclosed in a transparent plastic case. Individual blocks can be connected together via magnetic joints to form complex shapes. The position of each magnet is sensed by a Hall sensor grid attached on the back of the display platforms. A similar approach has been implemented in Pico [31].

Recently, alternative techniques for finger tracking were proposed (such as those relying on acoustic, radio waves and force sensing data). Hence, we can expect the availability of tangibles exploiting such technologies as soon as they will become mainstream.

### *B. Software toolkits for tangible-augmented interaction*

Various toolkits have been explicitly developed to simplify the creation of applications based on MTI and TUI. Most of them have been designed for devices relying on optical techniques and they are all centered on the idea of allow developers to focus on the design of their applications by abstracting (i) the underlying hardware level, and (ii) the complexity of the core interaction functionality, i.e. touch and fiducial identification and tracking.

The available toolkits can be broadly divided into two categories. The first includes libraries that merely deal with the low level input processing tasks, such as image segmentation, object recognition, noise filtering and calibration. Examples are reacTIVision [30], LightTracker [23], Touchlib [26] and Community Core Vision [24]. These toolkits usually run as standalone modules which encode and send data to an external application. The TUIO protocol [30] has become the standard de-facto for transmitting such interaction information.

The second category of toolkits embraces higher level frameworks, often built on top of libraries of the previous class. PyMT [25] is a Python library that supports several multi-touch gestures and multi-touch GUI. However, it cannot handle tangible interaction. Similar characteristics are offered by uTableSDK [29] and DiamondSpin [28]. Tactive [36] is another library for the development of MTI applications only. The advantage of Tactive is that it is based on a Javascript API that allows developer to use only web technologies and, consequently, to reuse available web applications in a multi-touch context. More advanced features are provided by TUIO AS3 [27], an Actionscript 3 (AS3) wrapper of the TUIO protocol that support tangibles. ToyVision [19] is another AS3 wrapper based on reacTIVision toolkit which extends the set of tangible

playing pieces available by including shape and color clues in the recognition process. Another contribution is provided by the ROSS library, which provides an abstraction of tangible platforms, full-body interaction spaces and responsive objects (e.g. RFID tags, smartphones), thus allowing a larger portion of the TUI space to be exploited into any ROSS based application. To the best of our knowledge, the only toolkit which is multiplatform and supports different recognition technologies is GestureWorks [34], a commercial framework which includes a large library of pre-defined gestures and offers as well a gesture markup language for defining new ones.

According to this taxonomy, the GAINE framework discussed in this paper falls into the higher level toolkits category, further extending the simplifications of the development process offered by other solutions. As a matter of facts, our frameworks offers (i) the management of both MTI and TUI interaction, (ii) independence from the computing platform, (iii) independence from the technology used to acquire interaction information, and (iv) high level constructs wich helps reducing the implementation burden.

## III. The GAINE Framework

As we stated in the introduction, GAINE is a software framework that supports rapid prototyping and development of applications based on tangible interaction. We stress the fact that GAINE, rather than being a general-purpose library, has been specifically designed for the development of projects with entertainment and edutainment purposes. In the following we will therefore refer to any GAINE-based application as “the game”.

The framework is capable of handling different hardware setups, from interactive tables to tablets, and is portable on different computing platforms. In the preliminary version of this work [41], GAINE had been designed to target interactive tabletops only, which are usually managed by desktop computers, running Windows, Linux or MacOS operative systems, and the framework development had been based on several portable Open Source C++ software libraries. However, on devices running iOS or Android, the use of such OS libraries raises several portability and integration problems. To overcome this issue, we ported GAINE into Unity 3D, a cross-platform game engine that supports a “write once – deploy everywhere” model. This means that an application based on Unity can be developed on any device and then deployed, with minimal effort, in any of the supported platforms, which include mobile devices, web browsers, desktops and consoles. This is possible since the game engine scripting language is built upon Mono [32], an Open Source and cross-platform porting of the Microsoft .NET framework. Another relevant characteristic of Unity is that it represents an “all-in-one” solution for the development of complex interactive graphical applications. Besides providing advanced lighting and rendering options, Unity includes built-in support for spatialized audio, physics management, complex animations, multitasking, pipeline optimization and networking, thus also allowing an extension of the features offered by the preliminary version of GAINE [41].

As a general overview, the GAINE framework is composed by different intercommunicating functional blocks (Fig. 1). The GameManager starts the application, reads the configuration scripts, stores and evolves

the application state and, finally, generates the outputs. The ObjectManager is responsible for handling the creation, destruction and state evolution of the digital playing pieces and of the application GUI. The EventManager is responsible for handling and communicating the events generated by both the users and the application. In the following subsections, we provide a detailed description of all these modules.

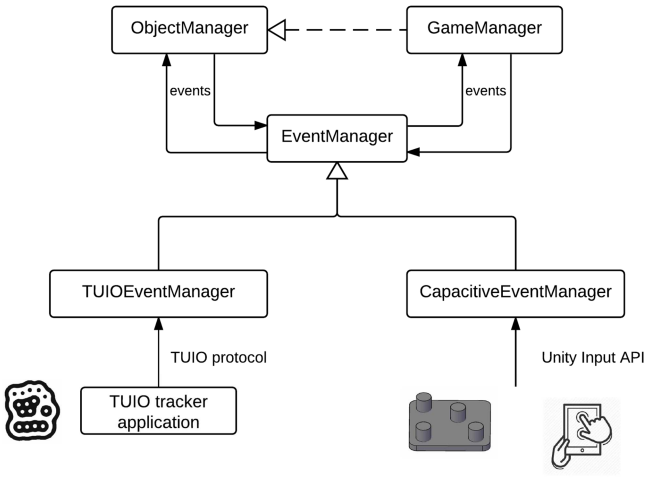


Fig. 1. GAINE module diagram

### A. *EventManager*

The EventManager implements a classic delegate event model, characterized by event sources (controls) and event listeners (consumers) that receive the events from the sources. As will be detailed in Section III.C, application designers can define handlers to specify the actions to be executed by the event receivers.

The main event source of any GAINE application is the tangible user interface, which provides data related to tracked fingers and tangible playing pieces. Since different target platforms offer different input methods and TUI data can be acquired with a plethora of technologies, from optical to capacitive or magnetic-based systems, the EventManager module has been designed to provide an abstraction of the hardware level. This objective has been achieved by implementing the basic event management functionalities into an abstract superclass, named EventManager, and the actual management of the tangible interaction into a set of subclasses each interfacing with a specific setup. In our current implementation we took into consideration two different configurations, namely interactive tables relying on optical techniques and using tangible objects identified by fiducial markers, and tablets featuring capacitive displays, where finger data are natively available and tangible objects are tracked through custom capacitive markers.

Regardless of the way input data are obtained, MTI data are first translated by the base class into multitouch gestures. Currently, only *tap*, *drag*, *pinch* and *rotate* gestures have been considered but, as future work, we are planning to handle other core gestures [20]. The three events generated by tangible objects are *add*, *remove* and *transform* (move or rotate). As for system events, user-defined events and those generated by the GUI elements, the EventManager class simply acts as a collector, pairing controls and consumers.

In the following, we will describe the implementation of the classes that interface interactive tabletops and tablets with GAINE.

*1) Interfacing interactive tables (TUIOEventManager)*

We developed a custom multitouch interactive table exploiting optical technologies and following several similar approaches in the literature [54][14][15][17][18]. In our table we use a combination of DI and FTIR, which provides a more robust result since DI can reliably track the fiducials, as long as the surface is equally illuminated, and FTIR offers a better solution for tracking fast finger movements. The table surface features a rear-projection 1080p screen. The height of the table is 110 cm and the screen size is 100x80 cm. The table can be connected with a secondary output, i.e. an external monitor or a projector, to provide either an information screen (e.g. for displaying game statistics) or a different view of the game environment (Fig. 2).

Fiducial and finger tracking is performed by the reacTIVision toolkit [53], which runs as a stand-alone module that streams captured data to the application software according to the TUIO protocol. These data include position and ID of detected fingers and position, orientation, size, ID and class type (i.e. the symbol drawn) of detected fiducials. The combination of ID and class type allows multiple instances of the same fiducial to be tracked. Since this module receives as input pure TUIO interaction data, it can be interfaced as is to any interactive device for which a module capable of streaming TUIO data is available.

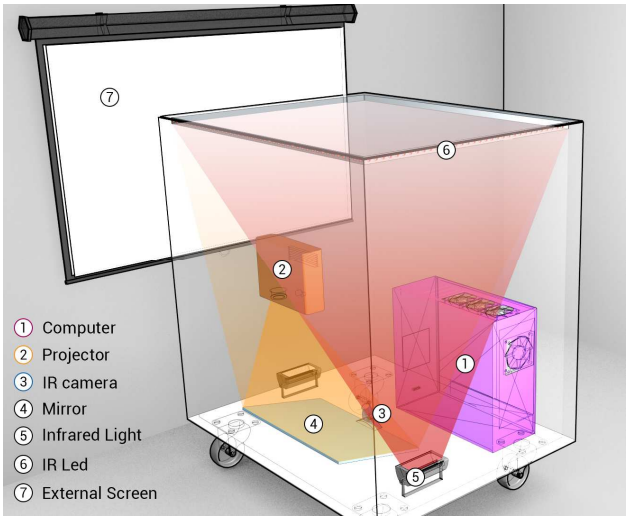


Fig. 2. A schematic of our multitouch interactive table.

*2) Interfacing Tablets (CapacitiveEventManager)*

In order to enable tangible interaction with tablets or devices equipped used with commercial capacitive touch screens, we developed custom markers following an approach similar to [46][47][48], i.e. exploiting passive markers characterized by unique patterns of conductive touch points that encode both their position and ID. In particular, since usual tablets' screen sizes require the smallest markers to avoid screen occlusion, our design was aimed at reducing as much as possible the marker size, which was the main limitation of previous works. Besides that, other constraints were taken into consideration. First, most capacitive screens limit to ten the number of detectable touches. Second, we experimentally found that a single contact point can be reliably detected on different devices if its size is greater than 6 mm and that two contact points can be told

apart if the distance between their centers is greater than 12 mm. All these constraints can be summarized in these two conflicting design objectives: (i) minimize the number of contact points per marker while maximizing the number of unique IDs, and (ii) minimize the marker size while guaranteeing a reliable extraction of the required information (marker position, orientation and ID).

The solution we proposed was using four contact points per marker, where three of them define an orthogonal Cartesian reference system capable of providing position and orientation information. The fourth one, the data point, defines the marker ID through its quantized position, on a regular grid of size 4 mm, in the marker reference frame. As a result, the minimal size that allows for robust marker identification is 30 mm. With this size, the number of unique IDs that can be represented is 8, and a larger set of distinct markers can be obtained increasing the marker size (see Fig. 3). Since recognizing a marker requires four touch points, a maximum of two markers and two finger touches can be recognized contemporarily on a standard tablet. For the construction of the markers we took advantage of the recent availability of conductive graphene filaments [52] on 3D printers, which were used to create the contact points, attached to a common base and enclosed in a plastic PLA shield (Fig. 4).

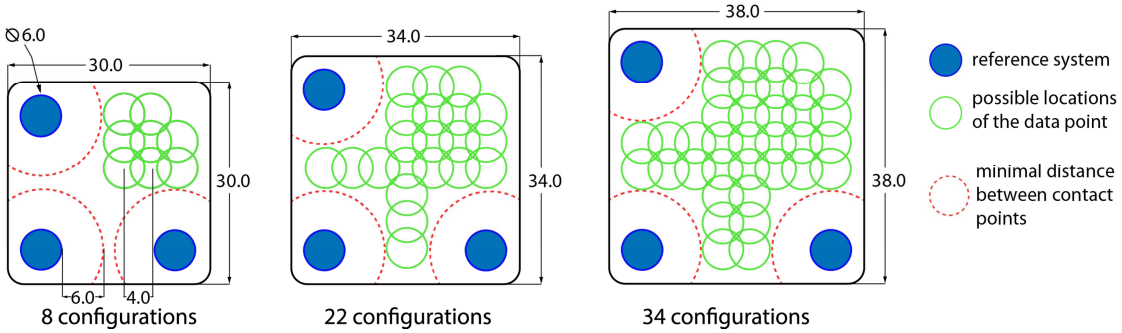


Fig. 3. Different marker sizes and their corresponding number of unique IDs

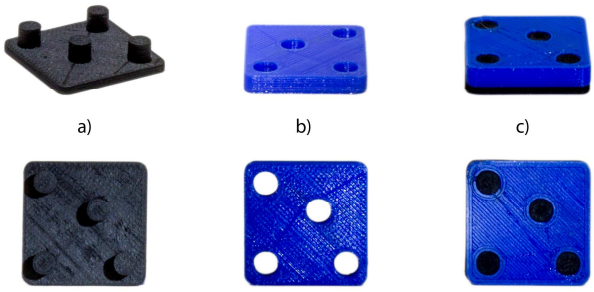


Fig. 4. The capacitive tangible (c) consists in a set of contact points (a) enclosed in a PLA shield (b)

The extraction of marker information from the touch points registered on screen is managed by a *marker engine*. Its main input source is the Unity Input API, which communicates the set of new, lifted or moving touches during the last frame. This set of data is then used for recognizing new markers and tracking or removing previously identified markers.

*Marker recognition*

New markers are detected by first identifying groups of four touch points whose geometric properties correspond to those defined by our markers. These points must satisfy the following properties: (i) three of

the points, $[\boldsymbol{V_1}, \boldsymbol{V_2}, \boldsymbol{V_3}]$, shall define two orthogonal vectors having a common vertex ($\boldsymbol{V_2}$) and lengths equal to $d$, the marker axis length[1], and (ii) the fourth point, $\boldsymbol{V_4}$, shall be included into the oriented bounding box of the first three points. Clearly, due to noise in the input data, the verification of these properties considers a suitable margin of tolerance. However, this noise affects as well the accuracy of all marker data. To improve the precision, we defined the following linear least square regression problem. Consider the ideal marker orthogonal reference system $[\boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{R_3}]$, with $\boldsymbol{R_2}$ placed in the origin and $\|\boldsymbol{R_1} - \boldsymbol{R_2}\| = \|\boldsymbol{R_2} - \boldsymbol{R_3}\| = d$. In absence of noise, the two reference systems $[\boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{R_3}]$ and $[\boldsymbol{V_1}, \boldsymbol{V_2}, \boldsymbol{V_3}]$ are related by the following transformation:

$$[\boldsymbol{V_1}, \boldsymbol{V_2}, \boldsymbol{V_3}] = M * [\boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{R_3}]^{\boldsymbol{t}} \qquad \text{(Eq. 1)}$$

where:

$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x_t \\ \sin(\theta) & \cos(\theta) & y_t \\ 0 & 0 & 1 \end{bmatrix}$$

is a roto-translational matrix defined by a rotation $\theta$ and a translation $(x_t, y_t)$. Due to noise, $M$ cannot be obtained directly, but it can be computed as the matrix that minimizes the sum of squared residuals between the model ($R$ points) and the observation ($V$ points):

$$\min_M \sum_i \left\| \boldsymbol{M} * \boldsymbol{R_i^t} - \boldsymbol{V_i} \right\|^2 \quad \text{(Eq. 2)}$$

This problem has the following closed-form solution. For each pair $(\boldsymbol{V_i}, \boldsymbol{R_i})$ of points, equation (1) can be expanded as:

$$V_{ix} = \cos(\theta) * R_{ix} - sin(\theta) * R_{iy} + x_t$$

$$V_{iy} = sin(\theta) * R_{ix} + \cos(\theta) * R_{iy} + y_t$$

and the whole set of equations can be summarized in matrix form as $A \cdot x = b$, where:

$$A = \begin{bmatrix} R_{1x} & -R_{1y} & 1 & 0 \\ R_{1y} & R_{1x} & 0 & 1 \\ R_{2x} & -R_{2y} & 1 & 0 \\ R_{2y} & R_{2x} & 0 & 1 \\ R_{3x} & -R_{3y} & 1 & 0 \\ R_{3y} & R_{3x} & 0 & 1 \end{bmatrix}, \quad x = \begin{bmatrix} \cos(\theta) \\ sin(\theta) \\ x_t \\ y_t \end{bmatrix}, \quad b = \begin{bmatrix} V_{1x} \\ V_{1y} \\ V_{2x} \\ V_{2y} \\ V_{3x} \\ V_{3y} \end{bmatrix}$$

Thus, the resulting solution of the minimization problem in equation (2) is:

$$x^* = (A^T \cdot A)^{-1} \cdot (A^T \cdot b)$$

Once we have $x^*$, we can update the values of $[\boldsymbol{V_1}, \boldsymbol{V_2}, \boldsymbol{V_3}]$, and, consequently, the marker data (position, orientation and its ID, which is obtained by projecting the data point $\boldsymbol{V_4}$ on the regular grid defined by $[\boldsymbol{V_1}, \boldsymbol{V_2}, \boldsymbol{V_3}]$). Once a new marker is recognized, it receives a unique identifier that, paired with the marker ID, allows again different instances of the same marker to be used at the same time.

*Marker tracking and removal and Finger tracking*

Markers are tracked exploiting the motion information of their touch points. After tracking, we apply again the same registration procedure previously described, this time exploiting as well the position of the data point $\boldsymbol{V_4}$. When one or more of the marker touch points are flagged as lifted, an event is raised signaling the

[1] Pixel to metric conversion relies on the device DPI

marker removal. Finally, the input touches not assigned to a new or tracked marker, are treated as finger touches and managed consequently by the base EventManager class.

### B. *Object Manager*

This module is responsible for creating, evolving and destroying all the digital contents of the game (i.e. the playing pieces and the GUI elements). These contents can be represented graphically as either 3D or 2D elements and they can be divided into four different classes, detailed in the following subsections: *game objects*, *tiles*, *agents* and *widgets*. These digital elements are defined in two different reference systems: the screen space for widgets and the world space for game objects, tiles and agents. The transformation that relates the two reference systems is managed directly by Unity according to the current settings of the virtual viewing camera.

We recall that a game deployed on our interactive table has the added benefit of an optional second screen. We will show at the end of this section that this feature is available for other devices and other platforms as well. The two screens are totally independent, and the developers can choose which contents to show and how they should be displayed on each of them.

#### 1) *Game objects*

Game objects are the basic digital elements managed and displayed by the game. Examples of game objects are the game board on the tabletop surface, the background 3D scenario shown on the secondary screen and interactive digital playing pieces. The developers can define all the characteristics of the classes of game objects in the configuration scripts. The main feature of a game object is its graphic representation. One or multiple 2D or 3D texturized models, called *switches* can be associated to each class. Each switch can be static or animated, and its animation can be controlled indipendentely (i.e. it can be started when an object is instantiated or when a specific event occurs, it can be lopped or not, and so on). The switch can also include a particle system which allows effects like smoke, fire, liquids or (why not?) magic spells to be simulated. The initial switch of a class instance can be chosen in different ways (e.g. randomly or according to a user-defined condition) and the actual switch can be changed according to events and game rules.

Game objects can be made interactive and the developers can define a handler for each of the touch events involving the object (see Section III.A). As an example, in a children's game, a geometric shape can change its color when the user taps on it, and it can be resized by pinching it.

#### 2) *Tiles*

The *tiles* are the digital elements representing the tangible game pieces. Each tile class is associated to a specific marker ID and is characterized by several parameters, most of which are inherited by the game objects. As for the tile switches, they can have any size, both related or not to the physical size of the corresponding tangible playing piece. The position and orientation of the tracked tangible are automatically

used to translate and rotate the tile representation in world space. Having a switch for a tile is not mandatory and, in the case of a double screen, the switch can also be displayed on one or both the outputs. For instance, as we will show in Section IV, tangibles placed on the tabletop can be the building blocks of a 3D city whose interactive navigation can be shown on the second screen.

Tile classes can also be defined as *groupable*, i.e. classes whose instances can be chained together to form a compound digital model. Examples of groupable tiles can be road parts or single houses forming a city block. In order to allow a correct match between the physical layout of the tangible objects and the digital representation of the tile group, groupable classes must be associated to (i) square markers and (ii) tile switches having a base with the same geometry and size of the marker. Two instances of a groupable class are connected when the distance between their centers is lower than a threshold related to the tile size. Grouped tiles are then linked to construct a 4 connected-neighbor graph and each group element is labeled according to the number and position of its neighbors as (Fig. 5): *Start* (tile with a single neighbor), *Linear* (two neighbors on the same grid direction), *Curve* (two neighbors on different grid directions), *TShape* (3 neighbors) and *Cross* (4 neighbors). A different switch can be defined for each label and the actual switch of an instance is instantaneously updated at every label change, i.e. when a tile is added or removed from the group.

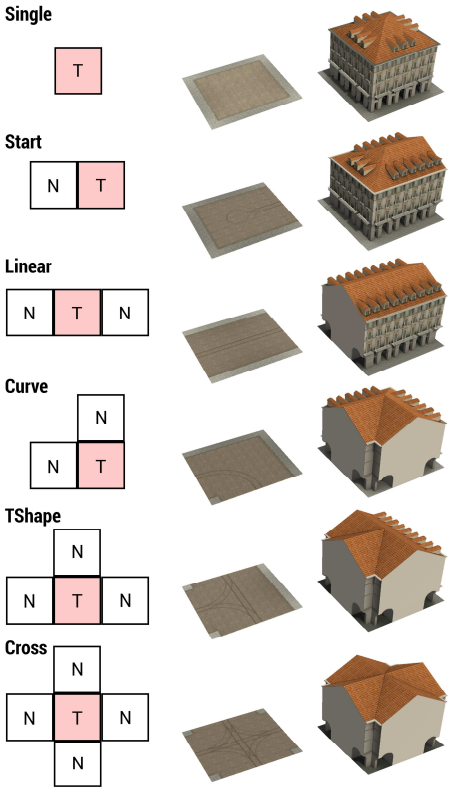


Fig. 5. Labels assigned to a grouped tile according to the number and position of neighboring Tiles (left column) and examples of label switches for two tile classes (Road, central column, and House, right column).

For groupable tiles, it is also possible to guarantee a seamless junction between graphical elements, even if tangibles are not perfectly aligned and connected. This is obtained by first creating an ideal grid matching the graph connections and aligned with the reference axes. This regularized grid is then used to update the position and orientation of the grouped tiles with a regression approach similar to the one described in section

III.A.2). This process, if enabled, is applied every time a tile group has been modified. An example of grid regularization is shown in Fig. 6.

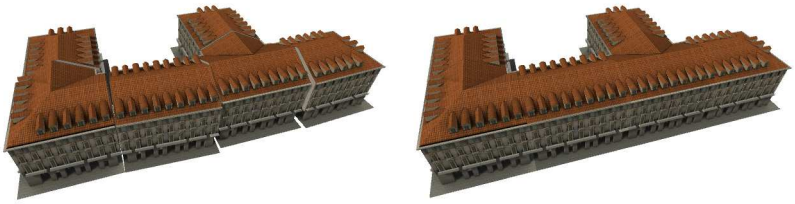

Fig. 6. An example of a tile group before (up) and after (down) grid regularization.

*3) Agents*

The *agents* are autonomous virtual characters that can be generated by the main scene (i.e. the game board or the 3D scenario on the secondary screen) or by digital (game objects) and tangible (tiles) playing pieces. In the current implementation, the agents' artificial intelligence is limited to provide them with the capability to navigate the environment in a "life-like and improvisional matter", relying on the so-called "steering behaviors" [4]. Actually, we offer an extension of the basic steering behaviors available in UnitySteer, the Unity porting of OpenSteer [33], and new type of behaviors can be easily added according to the needs of the application designer. Examples of basic steering behaviors are wandering around the world or inside a predefined area, move towards a static or dynamic target, follow a path and avoid obstacles. Complex behaviors can be obtained by combining the basic ones, i.e. following a leader avoiding obstacles or modeling coordinated motion of groups of agents.

Different behaviors can be assigned to an agent and, eventually, the agent behavior can be modified by events/conditions. An example game is shown in Fig. 7, where in a city scenario the house blocks generate as agents the city dwellers, which are then free to move around the city, and streets generate carriages, which are constrained to move on the correct street lane, and a tramway, which follows a predefined circular path.

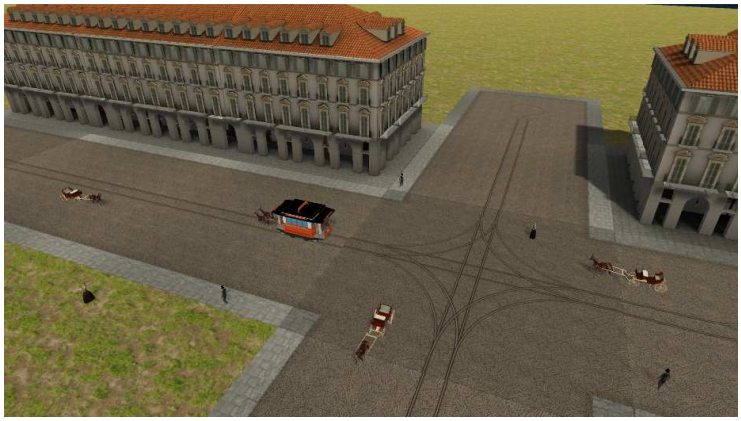

Fig. 7. Examples of autonomous agents generated by the playing pieces.

The agent characteristics can be again defined by the developers. Each agent can be represented by multiple animated switches. The minimal/maximal number of instances per class and the agent generation

rules and rates can be set in the scripts. Agents can be optionally destroyed according to their life-time, as a consequence of a system event or when the generating tile/tile group has been removed from the interaction surface. Other parameters associated to agents are their maximal speed, the list of obstacles to avoid during navigation (e.g. a list of game object, tile and agent classes) and a set of values to control their animation. When agents are generated by a groupable tile, the developer can also decide the minimal group size that allows the generation of agents, i.e. to avoid adding cars to a street composed by only two blocks.

*4) Widgets*

The widgets are the components of the game GUI (e.g. buttons, sliders, labels, image and video widgets). The various widget classes offer different pre-defined properties and generate events upon user interaction. The framework includes a peculiar type of widgets, called Sensible Areas, which are (invisible) polygonal regions defined on the interaction surface that raise an event when a marker moves within, enters or exit their area. The event data (i.e., position, instance and class ID of the marker that generated the event), can then be used by the application to trigger specific actions, as we will show in the examples in Section IV.

*5) Game view control*

GAINE provides a dual screen setup, which is natively available for our interactive table and can be added as feature on other devices as well. To manage the second screen, we implemented a client-server architecture, where the server is the (primary) interaction screen and the client is the secondary output. State synchronization between client and server is handled by the Unity networking API in a transparent manner for the developer. In this way, interactive tables where an external output is not available or devices like tablets can feature a second screen as well by simply running client and server on two separate hosts.

The game view on the main screen or on both displays can be controlled by the developers. Each display can define its own background (e.g. a 2D image/model for the game board on the tabletop surface and a 3D environment for the secondary screen) and a camera, which can provide a 2D or a 3D view, either parallel or perspective. All the camera parameters (position, orientation and field of view) can be modified during the game and multiple cameras can be defined in the scripts and switched at run time according to events or conditions. In a similar way, it is possible to define multiple lights, activate/deactivate them and modify their parameters. Game objects, tiles, agents and widgets can be displayed, according to the developers' choices, on one or both the display screens. For instance, in an augmented chess game deployed into the interactive table, the tabletop surface can display both a chessboard and the GUI elements to control the game. The tangible chess pieces can be represented as 3D models on a 3D virtual chessboard in the secondary display, allowing the game audience a better comprehension of the game evolution (Fig. 8).


Fig. 8. A chess game developed with GAINE: the playing pieces and their associated markers (left) and a snapshot of the game view displayed on the secondary screen (right).

### C. *GameManager*

Game management and evolution follows the event-driven paradigm, in which the execution flow is determined by *events* and it is controlled by *parameters* and *rules*.

*Events*, handled by the EventManager module (Section III.A), can be raised by user actions (i.e., tangible interaction events), by the system (e.g., alerts or recurring events generated by the system clock) or defined by the developers and associated to any GAINE element. For instance, an event can be generated by a tile class when the number of instances is equal, exceeds or drops below a predefined threshold or when a user-defined parameter has reached a certain value. Any GAINE element, including the system, can register to an event defining a handler, which consists in a set of rules that are executed only when the event is received.

*Parameters* are variables associated to the various application elements. Several parameters are already pre-defined in GAINE, such as the system date and time or the execution time. Standard parameters associated to tile and game object instances are their position, orientation and current switch. Class parameters, such as the number of created instances, are available as well for game objects, tiles and agents. Furthermore, developers have the possibility to create their own parameters and associate them to any element of the application. These parameters can also be defined as a function of other parameters. Any application element using a parameter (i.e. a condition to be checked or a function to be evaluated, a GUI label showing its value, and so on) is immediately notified when the parameter value has been modified.

The *rules* are sequences of actions that modify the game or an object state. Examples of such actions are an update of one or more parameters, the raise of an event and the play of a sound or the creation/destruction of different elements (e.g. a game object, a message window, a video widget, and so on). Rules can be stand-alone or associated to an event. Stand-alone rules are always associated with a conditional statement involving different parameters and are executed only if a parameter in the condition has been updated. Stand-alone rules are associated with a priority, an integer value defaulted to one that can be updated by the developer, and executed in descending priority order. In case of events, an event handler is defined as a set of rules, where each rule can be associated with an optional condition and a priority. Event rules are activated

when the event has been notified and only if the possible condition is verified. In other words, if one of the condition parameters has been updated but the event has not been raised, these rules are not executed.

In the Appendix, we report a detailed example showing how these elements can be combined in the implementation of a GAINE based interactive game.

## IV. Case Studies and Discussion

The GAINE framework has been used to develop several entertainment/edutainment applications, ranging from simple games to interactive stories for children. Some of these applications have been briefly sketched in the previous sections. Here we describe in more details several case studies and we report some data to show how the use of GAINE impacts the development of a multitouch, tangible augmented application.

### *A. Tic-Tac-Toe*

The first example is a tangible augmented version of the classic Tic-tac-toe game. Tic-tac-toe is a pencil-and-paper game for two players which take turns marking a 3x3 grid with their own symbol (either a “X”, cross, or a “O”, naught). The player who first succeeds to place three symbols on a row, column or diagonal wins the match. If all marks have been placed without a winner, the game ends with a draw.

This case study has been chosen for three main reasons. First, it is simple enough to allow us to briefly discuss its whole implementation. Second, it has been used to record some metrics to show the effectiveness of GAINE in supporting a rapid application development. Finally, it demonstrates how the same application can be ported, with a minimal effort, on both our interactive table and an Android tablet with a dual screen setup (Fig. 9). In the Appendix, we also present some excerpts of the scripts of this application to illustrate how the different GAINE elements are combined into the implementation.

In the tangible-augmented digital version of the game, the Tic-Tac-Toe board shows the 3x3 playing grid (Fig. 9(a)). The tangible playing pieces are the noughts and crosses, which are marked with two different markers (Fig. 9(b)). Their corresponding 3D tiles are shown on the secondary screen (Fig. 9(c)) together with a 3D perspective view of the game board. Each cell of the digital playing grid is associated with a sensible area that updates the cell value when a tangible is placed on or removed from the cell and raises the corresponding event. These events are then used to check for possible wins and eventually for concluding the game with a draw. When the game is finished, the players’ scores are updated and the new game button is enabled to start a new game.

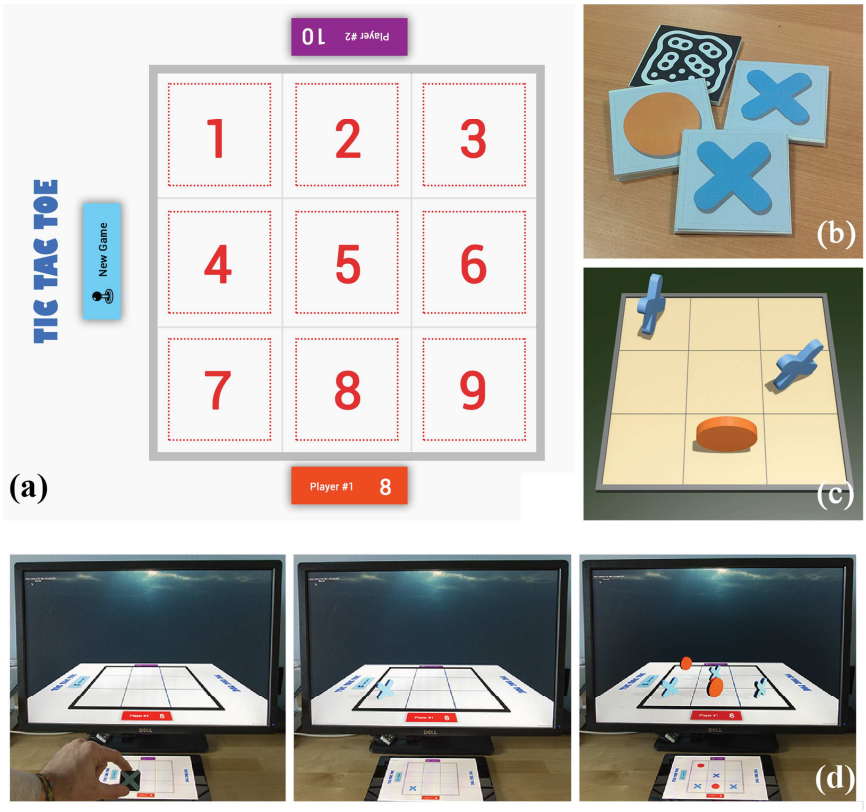


Fig. 9. (a) The Tic-tac-toe tabletop interface, showing the playing grid (where the dotted numbered squares highlight the sensible areas, which are not actually visible), the GUI labels and the start button; (b) the tangible playing pieces; (c) the 3D game view on the secondary screen; (d) images of the tablet version of the game and of its second screen

The whole development process of the game (which includes digital content and tangible pieces creation, script writing, testing and debugging) required a total of five man-hours and the number of code lines in the scripts is 372. In order to provide an initial, although partial, indication of the capabilities of our framework to support a rapid application prototyping, we asked one of our Computer Science MT students, an expert user of Unity, to develop from scracth the multi-screen, tangible Tic-Tac-Toe game in Unity. For a fair comparison, we provided him with the EventManager classes and the digital contents to be used in the game. As a result, there was an increase of +172% in the number of code lines required to define the game logic and the game management, while the development time almost doubled.

### B. *Torino 150*

The second example discussed is “Torino 150”, an edutainment application designed for children aged between 6 and 13. Torino 150 has a gameplay similar to that of the well-known SimCity game ([38]). It is a cooperative game exploiting the interactive table to involve the players in the task of founding and developing their own version of the city of Torino in 1861, the year when the city became the capital of the newly proclaimed united Kingdom of Italy. The game is aimed at communicating to children a piece of the history of their own city. The digital contents and the information displayed are the legacy of ToViA (Torino Virtual Adventure), a Virtual Heritage project aimed at constructing a realistic and historically accurate 3D model of Torino in 186 according to information collected from various sources, such as photos of actual buildings or historical documents like maps, drawings and paintings (Fig. 10).

The tangible playing pieces represent the different buildings of the city, that is houses, markets, police and fireman stations, industries and streets. Some blocks, such as the theatre, cathedral and royal palace, can be unlocked only reaching certain scores in the game. The players' task is to reach the highest level of "welfare" for citizens, i.e. a proper balance among positive (e.g. presence of hospitals, markets, public transportation) and negative factors (e.g. criminality, pollution, taxes). The game has three selectable difficulty levels, which (directly) affect the level of taxations and (indirectly) many other system parameters, thus making more challenging to achieve the game goals.

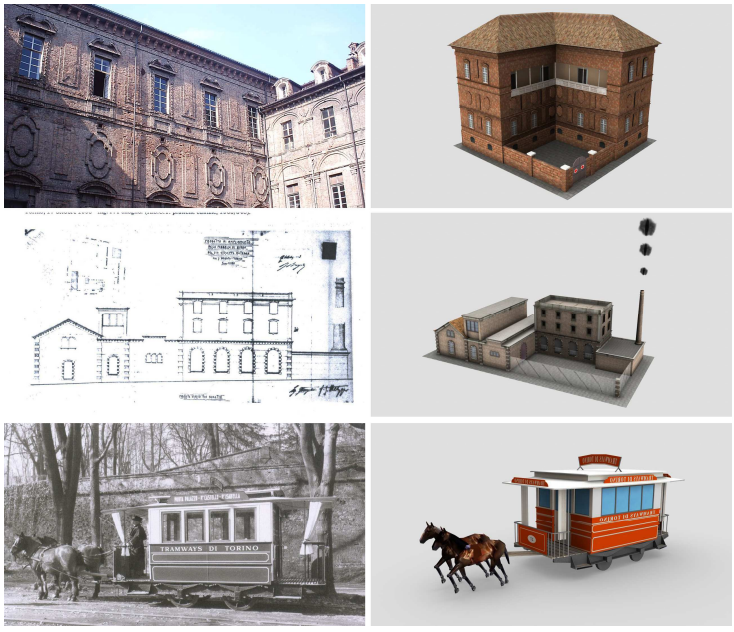


Fig. 10. Examples of models used by Torino 150 application and of reference documents used to create them.

The current value of the welfare and of other state parameters are computed as a function of the types and numbers of pieces placed on the table. Players need to find a proper balance between the played elements, since they can have a mutual influence. For instance, computation of welfare involves parameters like the population, the general health and richness and the level of criminality: the population increases with the number of houses, the health increases with the number of hospitals and decreases with the number of industries, the criminality is a function of population, richness and number of police stations, and so on. Thus, children need to discuss their choices in order to progress with the game.

This application exploits the second screen to immerse players in a 3D reconstruction of the virtual city they are building in the game (Fig. 11). This virtual environment can be observed by different perspectives or navigated either using an interactive camera or through a first-person camera attached to any of the citizens moving in the environment. The current viewing camera can be selected and updated through the game GUI.

The virtual city is populated by different autonomous agents. House blocks generate the city dwellers and the street blocks create horse-drawn trams and carriages (see Fig. 7). Some system events were also introduced. For instance, if the cathedral has been placed on the table, it rings the bell at predefined intervals to announce a religious function causing citizens to approach the church. As another example, we added an earthquake event, which can be raised either randomly by the system or explicitly by players with a GUI

button, which introduces additional challenges by causing the partial or total collapse of some houses and a consequent population decrease.

As for the development issues, the only modeling effort was the adaptation of 3D models developed for a previous project to the needs of a real-time VR application and the modeling of the different switches for the groupable tiles (houses and streets). The amount of code required by the application scripts to configure contents and implement the game logic sums up to 513 lines only. We think this value can provide another indication of the effectiveness of the framework.

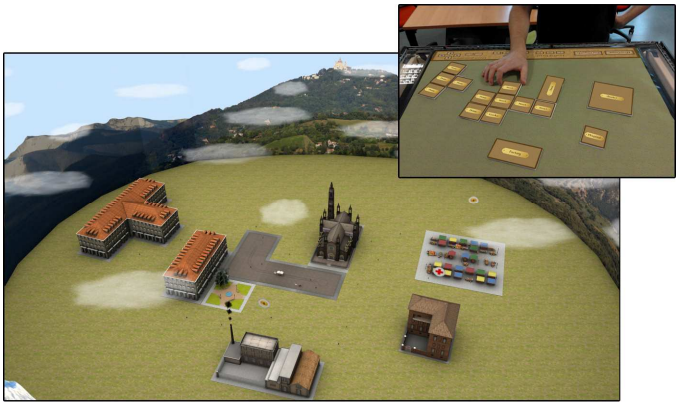

Fig. 11. Torino 150 games: the playing pieces and a screenshot of the view on the secondary screen.

### C. *Shapes*

Shapes is a tangible-augmented tablet version of the various Shape Matching games aimed at helping children to familiarize with the basic properties of simple geometric figures. By figuring out how to drop each piece into its proper hole, children learn to categorize and eventually name shapes. The capability to exploit physical tokens on a tablet delivers a series of benefits from the traditional version of the game. First, it is possible to improve the user experience by adding various feedbacks related to the accomplishment of every matching task, such as sounds and visual effects. Second, from a game design perspective, it is possible to challenge players with more difficult task and puzzles than the mere matching.

For this application, we designed six tokens with different geometrical shapes: triangle, square, circle, rhombus, pentagon and hexagon. Players are required, on a number of levels of increasing complexity, to match position and orientation of a shape displayed on screen with that of the corresponding token or to create compound figures with the available tokens.

### D. *Colombo*

Colombo is a tablet based collaborative game for two players where the main objective is to navigate a sailboat along a river avoiding collisions with its banks. The screen is divided horizontally in the two areas where each player can place his/her blower, represented by a tangible playing piece (Fig. 12(a)). The forward direction of the wind generated by each blower is determined by the line connecting the marker and the boat centers and it spans the wind zone, an angle determined by the marker orientation. The module of the wind force is a function of both the marker-boat distance and of the angle the wind direction makes with the bisector of the wind zone (Fig. 12(b)). When both player tokens are placed, the resultant of the two

corresponding forces is applied to the boat (Fig. 12(c)). Thus, the game requires both communication and coordination between players to fulfill the objectives.

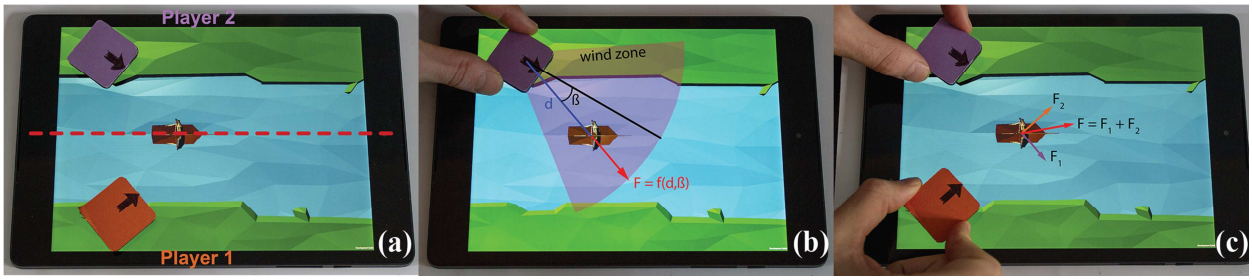


Fig. 12. Controlling the boat movements in Colombo

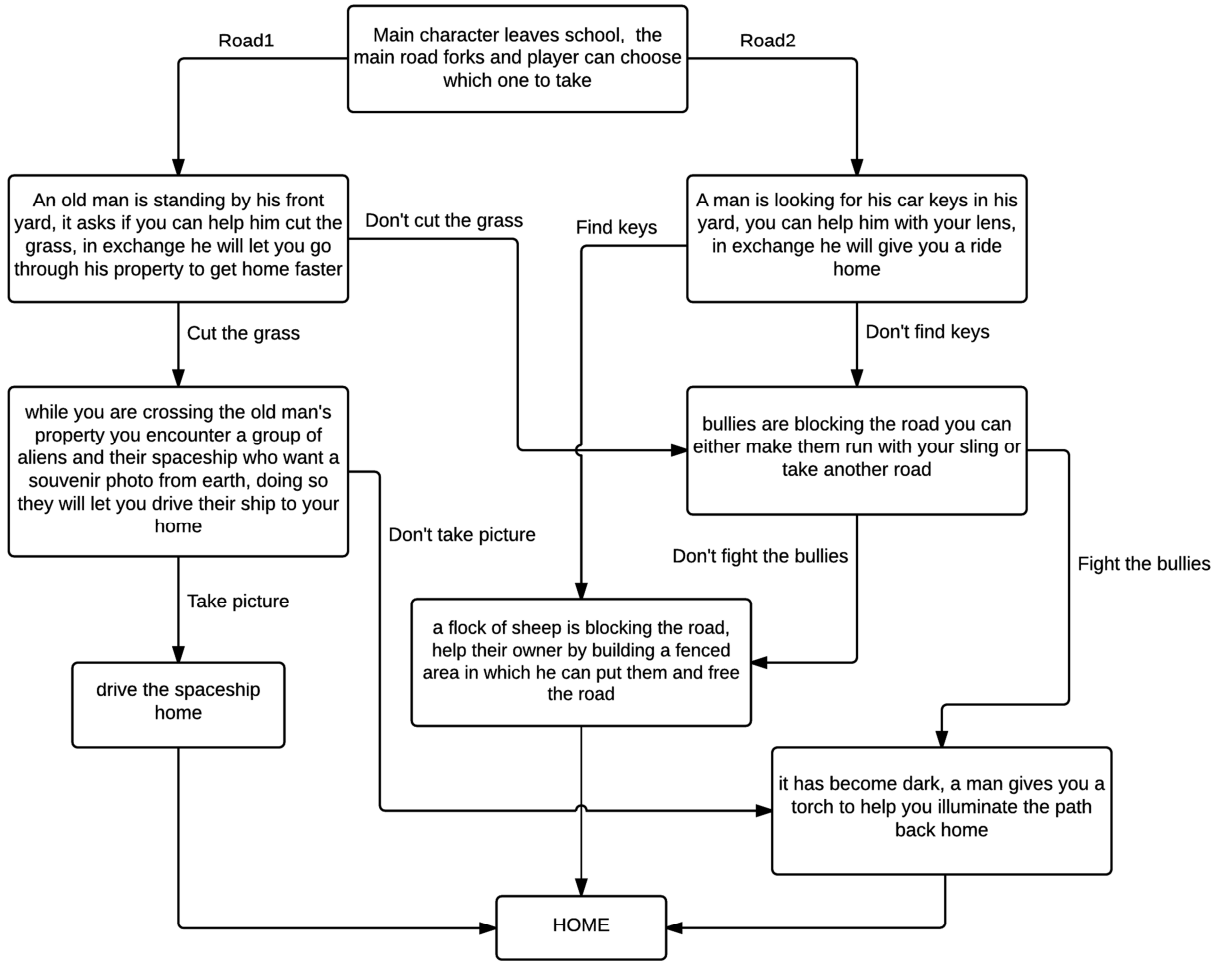


Fig. 13. Outline of *My Way Home* dynamic plot, where every block and every branch contribute to the evolution of the storyline

### *E. My Way Home: Interactive Storytelling*

Interactive storytelling is a form of narrative where the user is not bounded to a linear progressing storyline [3]. On the contrary, the reader can actively influence the narrative evolution, either by determining the main character's actions or by altering the environment in which the story takes place. To this end, the use of MTI and physical tokens offer a direct and tangible representation of the elements manipulated in the digital world and, thus, it can improve the general impact and immersion into the storyline dynamics ([9],[1]).

In order to show how GAINE can effectively support interactive storytelling, we implemented *My Way Home*, the story of a boy who misses the bus right after school and needs to find a way back home. On his journey, he will face hurdles to overcome and meet characters that can help him to fulfill his objective. The story evolves as a series of scenes in which every decision the player takes has an influence on the story progress (Fig. 13). The player can use tangibles to make choices, to control the main character and to exploit tools in achieving her/his goals. As an example, the story features a scene where an old man asks the player to help him find the car keys he lost in the front yard and the player can use a tangible-controlled "magnifying glass" as a helper tool (Fig. 14). As a reward for a successful search, the old man will kindly give the boy a ride home. Optionally, the player can choose to neglect his request and proceed on his/her own exploration of the environment.

Fig. 14. Examples of tangible used as tools to progress in the story (magnifying glass, left, torch, right)

## V. Conclusions

In this paper we presented GAINE, a flexible framework for the rapid prototyping and development of edutainment and entertainment applications based on multitouch and tangible interaction. The main contributions of GAINE are the following. First, it provides a set of high level constructs and a scripting language that allows both the characteristics of the digital contents and the game logic to be defined in a straightforward way, thus reducing the development time. Second, the framework is portable on different hardware and platforms, thus supporting a large number of different devices and different types of tangible objects. In the paper we provided several examples of edutainment applications developed with GAINE, showing its effectiveness in supporting the rapid prototyping of such tangible-augmented games. We underline the fact that researchers from different areas (e.g. usability engineers, pedagogists, museologists) were involved in the design of our games, which in turn gave us ideas to improve the list of available features. Hence, the development of our framework gained some indirect benefit from such multidisciplinary expertise.

As for future work, we are planning to expand the functionalities offered by GAINE. We are currently investigating the problem of extending the complexity of the AI offered by our agents. One option could be to integrate VIRTUAL-ME, a software library for Unity we developed in a previous project [35]. VIRTUAL-ME allows programmers to define and manage the human-like behaviour of autonomous characters in terms

of tasks, needs, emotions, perception and memory. We believe that such a feature would certainly extend the range of applications that can be developed with GAINE. As another feature, following the ideas discussed in [19] and [37], we are also planning to enhance the tangible interaction component by providing support for a more varied set of playing pieces and responsive objects. Finally, the design process would definitely benefit from an IDE (similar to the Graphic Assistant described in [19]) to define the characteristics of the game elements and automate the script generation.

## APPENDIX

Here we show some excerpts of the scripts implementing the Tic-Tac-Toe tabletop game described in Section IV.A. In order to manage the game, the following system parameters and events were defined:

```
<parameters>
  <parameter  name="winPlayer1" type="int" value="0"/>   <!-- player1 = cross -->
  <parameter  name="winPlayer2" type="int" value="0"/>   <!-- player2 = nought -->
  <parameter  name="gameActive" type="bool" value="true"/>
  <parameter  name="gameResult" type="int" value="-1"/> <!--  0: draw, "cross": player1, "nought": player2 -->
  <parameter  name="grid[9]" type="int" value="{0,0,0, 0,0,0, 0,0,0}"/>
  <parameter  name="playedPieces" type="int" value="0"/>
</parameters>
<eventList> <event name="OnGameFinished"/> </eventList>
```

`Grid` is a vector of nine elements that records the playing pieces placed on the grid cells (Fig. 9(a)), whose number is stored in `playedPieces`. `GameResult` stores the result of the current game (0 for a draw or the marker ID of the pieces played by the winner), `winPlayerX` record the players' scores and `gameActive` indicates when a game is in progress. The two tile classes, cross and naught, are configured as follows:

```
<tile type="Nought" marker="105" />
<switch condition="gameResult == marker" model="models/nought_ani.unity3d" animated="true" loop="true" display="2"/>
<switch condition="gameResult != marker" model="models/nought.unity3d" display="2"/> </tile>
<tile type="Cross" marker="205" />
<switch condition="gameResult == marker" model="models/cross_ani.unity3d" animated="true" loop="true" display="2"/>
<switch condition="gameResult != marker" model="models/cross.unity3d" display="2"/> </tile>
```

Both tiles have two switches displayed on the secondary screen, a static one when the game is running and an animated one when the game ends with a win of the corresponding player. The current switch is determined by a condition on `gameResult`. Different system rules are used to check for wins and draws:

```
<ruleList>
  <!-- check possible win in the first grid row -->
  <rule condition="gameActive and grid[1] == grid[2] and grid[2] == grid[3] and grid[3] != 0">
   <onTrue action="gameActive = false, gameResult = grid[3]" emitEvent="OnGameFinished" /> </rule>
<!– we do not report, for the sake of brevity, the similar rules that check different combinations-->
<!– draw condition (this rule has a lower priority check first possible wins and eventually signal a draw) -->
  <rule condition="playedPieces == 9" priority="0">
    <onTrue  action = "gameActive = false, gameResult = 0" emitEvent ="OnGameFinished" /> </rule>
</ruleList>
```

Each rule is guarded by a condition (`condition` field) that, when verified (`onTrue` tag), can cause the execution of some actions (`action` field) and/or the raise of one or more events (`emitEvent` field). The rule conditions involve the grid values and the number of played pieces, which are updated when a marker is placed on (`OnSensibleAreaEntering` event) or removed from (`OnSensibleAreaLeaving` event) a sensible area. Each sensible area defines an extra parameter `cell`, the index of the grid cell it is associated with:

```
  <sensibleArea name="c1" vertices="(…)"> <parameter name="cell" type="int" value="1"> </sensibleArea>
<!– other cells from c2 to c8... -->
  <sensibleArea name="c9" vertices="(…)"> <parameter name="cell" type="int" value="9"> </sensibleArea>
```

The sensible area event handlers update the number of pieces played and assigns the proper entry of the `grid` vector to the class ID of the played piece (for a tangible placed on the table) or to zero (for a removed tangible). Eventually, a modification of these values causes the raise of the `OnGameFinished` event, which determines the update of the current scores and halts the game. When the "New game" button is pressed, the grid values are reset and a new game is started. The following handlers implement these behaviors:

```
<eventListeners>
  <event name="OnSensibleAreaEntering">
    <rule action="grid[sender.cell] = sender.marker, playedPieces = playedPieces + 1" /> </event>
  <event name="OnSensibleAreaLeaving">
    <rule action="grid[sender.cell] = 0, playedPieces = playedPieces - 1" />  </event>
  <event name="OnGameFinished">
    <rule action="gameActive = false, newGameButton.enable = true"/>
    <rule condition="gameResult == 0">
      <onTrue playSound="sounds/draw.wav"/> </rule>
    <rule condition="gameResult == Cross.marker">
      <onTrue action="winPlayer1 = winPlayer1 + 1" playSound="sounds/win1.wav"/> </rule>
    <rule condition="gameResult == Nought.marker">
      <onTrue action="winPlayer2 = winPlayer2 + 1"  playSound="sounds/win2.wav"/> </rule>
  </event>
  <event name="OnButtonClicked">
    <rule action="grid[] = {0,0,0, 0,0,0, 0,0,0}, gameResult = -1, gameActive = true,
       newGameButton.enable = false, playedPieces = 0" /> </event>
</eventListeners>
```

The widget configuration file[2] creates the labels displaying the players' scores and the "New Game" button, which is disabled at game start and raises, if active and clicked, a default event `OnButtonClicked`.

```
  <label name="PlayerOneLabel" … displayedParameter="winPlayer1" />
  <label name="PlayerTwoLabel" … displayedParameter="winPlayer2" />
  <button name="newGameButton" … enable="false" />
```

Summarizing, the game execution flow is the following: (i) when a symbol is placed on a grid cell, an event is raised causing an update of the cell value, an increment of the number of pieces played, and an evaluation of the rules including one of these parameters in their conditions; (ii) eventually, the `OnGameFinished` event is raised, halting the game, updating the scores and the labels showing them, playing a proper sound and activating the "New Game" button; the tiles of the possible winner switch to their animated version; (iii) when the "New Game" button is pressed, game variables are cleared, the button is deactivated and a new match can start.

The tablet version of the application has a different marker management. Since the device limits the use of a maximum of two tangibles at the same time, the markers act as molds to create, when placed on the playing grid, two gameObjects, a 2D one on the tablet screen and a 3D one on the second screen. Then, the OnSensibleAreaEntering event handler includes an extra condition to avoid placing a marker on a cell already occupied, and the OnSensibleAreaLeaving handler has not been implemented.

## REFERENCES


[1] Thijs Alofs; Mariët Theune; Ivo Swartjes. "A tabletop interactive storytelling system: designing for social interaction". International Journal of Arts and Technology (IJART), Vol. 8, No. 3, 2015. pp.188 - 211

[2] M.A. Evans and J. Rick. Supporting Learning with Interactive Surfaces and Spaces. In Handbook of Research on Educational Communications and Technology, 2014, pp 689-701

[3] Galyean T. "Narrative Guidance of Interactivity". MIT Ph.D. Thesis. 1995

[4] C. W. Reynolds, "Steering Behaviors For Autonomous Characters," In *Proc. of Game Developers Conference*, San Jose, California, 1999, pp. 763-782.


[2] For the sake of clarity, we report a simplified version of the script, where we removed most of the parameters related to the definition of the widget appearance

[5] A. Battocchi, A. Ben-Sasson, G. Esposito, E. Gal, F. Pianesi, D. Tomasini, P. Venuti, P. Weiss, and M. Zancanaro, "Collaborative puzzle game: a tabletop interface for fostering collaborative skills in children with autism spectrum disorders," *Journal of Assistive Technologies*, 2010, 4(1), pp. 4-13.
[6] G. F. M. Silva, A. Raposo and M. Suplino, "Par: A collaborative game for multitouch tabletop to support social interaction of users with autism," *Procedia Computer Science*, 27, 2014, pp. 84-93.
[7] R. Zarin and D. Fallman, "Through the troll forest: exploring tabletop interaction design for children with special cognitive needs," In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2011, pp. 3319-3322.
[8] C. McCrindle, E. Hornecker, A. Lingnau and J. Rick, "The design of t-vote: a tangible tabletop application supporting children's decision making," In *Proceedings of the 10th International Conference on Interaction Design and Children*, 2011, pp. 181-184.
[9] Ana Paiva. "The role of tangibles in interactive storytelling". In Proceedings of the Third international conference on Virtual Storytelling: using virtual reality technologies for storytelling (ICVS'05), 2005, 225-228
[10] Y. Li, W. Fontijn and P. Markopoulos, "A tangible tabletop game supporting therapy of children with cerebral palsy," *Proceedings of the 2nd Intl. Conf. on Fun and Games*, 2008 pp. 182-193.
[11] J. Duckworth, P. R. Thomas, D. Shum and P. H. Wilson, "Designing co-located tabletop interaction for rehabilitation of brain injury," *Lecture Notes in Computer Science*, Vol. 8013, 2013, pp 391-400.
[12] S. E. Higgins, E. Mercier, E. Burd and A. Hatch, "Multi-touch tables and the relationship with collaborative classroom pedagogies: A synthetic review," *International Journal of Computer-Supported Collaborative Learning*, 6(4), 2011, pp. 515-538.
[13] X. Cao, S.E. Lindley, J. Helmes, A. Sellen, "Telling the whole story: Anticipation, inspiration and reputation in a field deployment of TellTable," *Proceedings of CSCW 2010, ACM Conference on Computer Supported Cooperative Work.* p. 251-260.
[14] T. Alofs, M. Theune, and I.M.T. Swartjes, "A Tabletop Board Game Interface for Multi-User Interaction with a Storytelling System," *Proceedings of 4th International Conference on Intelligent Technologies for Interactive Entertainment, INTETAIN 2011*, pp. 123-128.
[15] A. Wu, D. Joyner and E.Y.L. Do, "Move, beam, and check! Imagineering tangible optical chess on an interactive tabletop display," *Computers in Entertainment (CIE)* 8, no. 3, 2010: 20.
[16] M. Horn, Z. Atrash Leong, F. Block, J. Diamond, E. M. Evans, B. Phillips and C. Shen, "Of BATs and APEs: an interactive tabletop game for natural history museums," In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, pp. 2059-2068.
[17] N. Correia, T. Mota, R. Nóbrega, L. Silva, A. Almeida, "A multi-touch tabletop for robust multimedia interaction in museums," In *ACM International Conference on Interactive Tabletops and Surfaces*, 2010 pp. 117-120,
[18] S. Jordà, G. Geiger, M. Alonso and M. Kaltenbrunner, "The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces," In *Proceedings of the 1st international conference on Tangible and embedded interaction*, 2007, pp. 139-146.
[19] J. Marco, E. Cerezo and S. Baldassarri, "ToyVision: a toolkit for prototyping tabletop tangible games," In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, 2012 pp. 71-80.
[20] C. Villamor, D. Willis, L.Wroblewski,"Touch Gesture Reference Guide. 2010. Available online at: static.lukew.com/TouchGestureGuide.pdf, last accessed December 2015
[21] N. Matsushita and J. Rekimoto, "HoloWall: designing a finger, hand, body, and object sensitive wall," In *Proceedings of the 10th annual ACM symposium on User interface software and technology (UIST '97)*, 1997 pp. 209-210.
[22] Y.J. Han, "Low-cost multi-touch sensing through frustrated total internal reflection," In *Proceedings of the 18th annual ACM symposium on User interface software and technology (UIST '05)*, 2005 pp. 115-118.
[23] A. Gokcezade, J. Leitner, and M. Haller, "LightTracker: An open-source multitouch toolkit," *ACM Comput. Entertain.* 2010, 8, 3, pp. 16.
[24] Community Core Vision, available online at github.com/nuigroup/ccv15, last accessed December 2015
[25] T. Hansen, C. Denter and M. Virbel, "Using the PyMT toolkit for HCI Research," *Forum on Tactile and Gestural interaction*, 2010.
[26] Touchlib, available online at nuigroup.com/touchlib, last accessed December 2015
[27] J. Luderschmidt, I. Bauer, N. Haubner, S. Lehmann, R. Dörner and U. Schwanecke, "TUIO AS3: A Multi-Touch and Tangible User Interface Rapid Prototype Toolkit for Tabletop Interaction," In *Self Integrating Systems for Better Living Environments: First Workshop, Sensyble*, 2010, pp. 21-28
[28] C. Shen, F. D. Vernier, C. Forlines and M. Ringel, "DiamondSpin: an extensible toolkit for around-the-table interaction," In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2004, pp. 167-174.
[29] uTableSDK. Available online at http://utablesdk.codeplex.com, last accessed December 2015
[30] ReacTIVision, a toolkit for tangible multi-touch surfaces, available online at http://reactivision.sourceforge.net/, last accessed December 2015
[31] Patten, J., and Ishii, H. Mechanical constraints as computational constraints in tabletop tangible interfaces. In Proc. CHI '07 (2007), 809–818
[32] Mono, available online at http://www.mono-project.com/, last accessed December 2015
[33] OpenSteer, Steering Behaviors for Autonomous Characters, available online at opensteer.sourceforge.net, last accessed December 2015
[34] GestureWorks, accessible online at http://gestureworks.com/, last accessed December 2015
[35] *Reference removed for complying with double bind review process constraints*
[36] O. Gaggi and M. Regazzo, "Tactive, a Framework for Cross Platform Development of Tabletop Applications," In *International Conference on Web Information Systems and Technologies (WEBIST 2014)*, pp. 91-98.
[37] A. Wu, J. Jog, S. Mendenhall, A. Mazalek, "A framework interweaving tangible objects, surfaces and spaces," In *Human-Computer Interaction. Interaction Techniques and Environments*, 2011, pp. 148-157.
[38] SimCity, from Wikipedia http://en.wikipedia.org/wiki/SimCity, last accessed December 2015
[39] Loureiro, B.; Rodrigues, R., "Multi-touch as a Natural User Interface for elders: A survey," in Information Systems and Technologies (CISTI), 2011 6th Iberian Conference on , vol., no., pp.1-6, 15-18 June 2011
[40] C. OMalley and D.S. Fraser, "Literature Review in Learning with Tangible Technologies," NESTA Futurelab Report, vol. 12, 2005
[41] *Reference removed for complying with double bind review process constraints*
[42] L. Chan, S. Müller, A. Roudaut, and P. Baudisch, "CapStones and ZebraWidgets: sensing stacks of building blocks, dials and sliders on capacitive touch screens," *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012. p. 2189-2192
[43] S. Kratz, T. Westermann, M. Rohs and G. Essl, "CapWidgets: tangile widgets versus multi-touch controls on mobile devices," In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2011. p. 1351-1356
[44] R. H. Liang, L. Chan, H. Y. Tseng, H. C. Kuo, D. Y. Huang, D. N. Yang and B. Y. Chen, "GaussBricks: magnetic building blocks for constructive tangible interactions on portable displays," In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014. p. 3153-3162
[45] R. H. Liang, H. C. Kuo, L. Chan, D. N. Yang and Bing-Yu Chen, "GaussStones: shielded magnetic tangibles for multi-token interactions on portable displays," *In Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 2014. p. 365-372.
[46] H. Schaper, "Physical Widgets on Capacitive Touch Displays," 2013. PhD Thesis. RWTH Aachen University
[47] S. Voelker, K. Nakajima, C. Thoresen, Y. Itoh, K.I. Øvergård and J. Borchers, "PUCs demo: detecting transparent, passive untouched capacitive widgets," In *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces*. ACM, 2013. p. 325-328.
[48] N. H. Yu, L. W. Chan, S. Y. Lau, S. S. Tsai, I. C. Hsiao, D. J. Tsai, F. I. Hsiao, L. P. Cheng, M. Chen, P. Huang and Y. P. Hung, "TUIC: enabling tangible interaction on capacitive multi-touch displays," In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*. ACM, 2011. p. 2995-3004
[49] Tiggly, Interactive Toys & IPad learning apps for childrens, available online at https://www.tiggly.com/, last accessed December 2015
[50] Ishii, H.. The tangible user interface and its evolution. Commun. ACM 51, 6, 2008, 32-36

[51] AppMATes, mobile application toys, http://www.appmatestoys.com/, last accessed December 2015
[52] Conductive Graphene Filament, a conductive 3D printing filament, Graphene Lab Inc., http://goo.gl/6ZJjqf, last accessed December 2015
[53] M. Kaltenbrunner, R. Bencina. "reacTIVision: a computer-vision framework for table-based tangible interaction". *In Proceedings of the 1st international conference on Tangible and embedded interaction (TEI '07)*, pp. 69-74
[54] T. Gross, M. Fetter and S. Liebsch, "The cuetable: cooperative and competitive multi-touch interaction on a tabletop," In *CHI 2008 Extended Abstracts on Human Factors in Computing Systems*, pp. 3465–3470.